SAND: 2012-1717C

Ideas for Future High Performance CPU and System Architectures

Dave Resnick
Sandia National Laboratories

Sandia is a Multiprogram Laboratory Operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy Under Contract DE-ACO4-94AL85000.

Starting Out: Today's Limits

- Board- and cabinet-level power
- Difficulty in parallelizing applications
- Low efficiencies and high overheads make for usage complexity and lower than desirable performance
- Memory and IO scalability: bandwidth and size
- High-overhead communication and data sharing
- Easier to create complexity to support old ways of doing things than to say "Enough!" and find ways forward that make breakthroughs while supporting current workloads

We are not in the 1970s. Our capabilities have increased by more than a million times. Requirements have increased even further. Why accept limits or ways of doing things established then?

What follows is a fairly spare outline of some basic ideas. There are lots of additional thoughts, insights, and additional details above what is presented here. Welcome NDAs, if needed, to support deeper technical interactions. Lets talk!

There are many possibilities—and opportunities—to establish roadmaps to reasonably (if not totally easily) move into the future, using the ideas presented here as goals and suggested new ways to do things

There needs to be general co-design at all levels to fully develop these ideas and to integrate with other ideas, requirements, and goals

Ideas for Moving Ahead

- Make systems memory-centered rather than CPU centered
- Put new functions into memory systems to support parallelism and execution efficiency; include NV memory
- Extend system addressing to cover all sources of processing capabilities: cores, *threads*, network functions, IO, *memory parts*… so can greatly increase efficiency and lower overhead
- Replace CPU chip caches with a more general memory that is internally intelligent and is controllable by user applications
- Make CPU implementations much more efficient with multi-thread cores and new internal organizations
- Make data sharing and communication (including coherency operations) much easier, and with low latency and reduced energy

Memory System

- Micron HMC (Hybrid Memory Cube) technology breaks through the memory wall (**100x** bandwidth per component!) and provides opportunities for new memory functions
  - Memory Atomics greatly improve some coherency operations
  - Memory Moves (including Gather/Scatter, Transpose, etc.) for increased parallelism and processing efficiency
  - Low-overhead resiliency functions (CubeKill, ModuleKill, auto-recovery operations), including recovery from path failures
  - Coherency and communication capabilities to make those functions easier, faster, and lower energy
  - Should be reasonable to include some processing capability
  - Easy memory scaling (to terabytes connected to a CPU with small energy impact for increased size—less than linear increase)
  - Bring non-volatile memory to up to 1st level integration and functionality with DRAM memory

Local Memory Store Replacing Caches

LMStr* eliminates cache issues and offers new functions

- Memory functions
  - Scratch
  - Cache
  - Message/data source and destination (no main memory copies)
  - Shared/Private as needed
- Application controlled—both for amount of memory per process and for functionality.
- Ability to 'go around' LMStr to directly address main memory (eliminate 'use once' references to cache so don't contaminate local memory still in use)
- Built-in error detection and auto-recovery. Memory failures automatically removed from operation (generally with no impact on users)

*Sandia Conceptual Design

Easy Communication: MCCU

Management, Communication, Coherency Unit*

Start by upgrading system addressing to include thread-level components

- Upgrade address capability so can send signal flags directly between processes.
- Include multiple 'signal blocks' in CPUs that remote processes can activate. Receiving processes can test signal state. Also, a signal going set can cause a branch to a handler function, if enabled. No interrupts, no polling for coherency!
- Some signal blocks used to determine when multiple other instructions (generally memory operations) are complete and so safe to continue. Example: Can indicate to another process that needed memory operations are complete and so safe for the receiving process to access.

*Sandia Conceptual Design

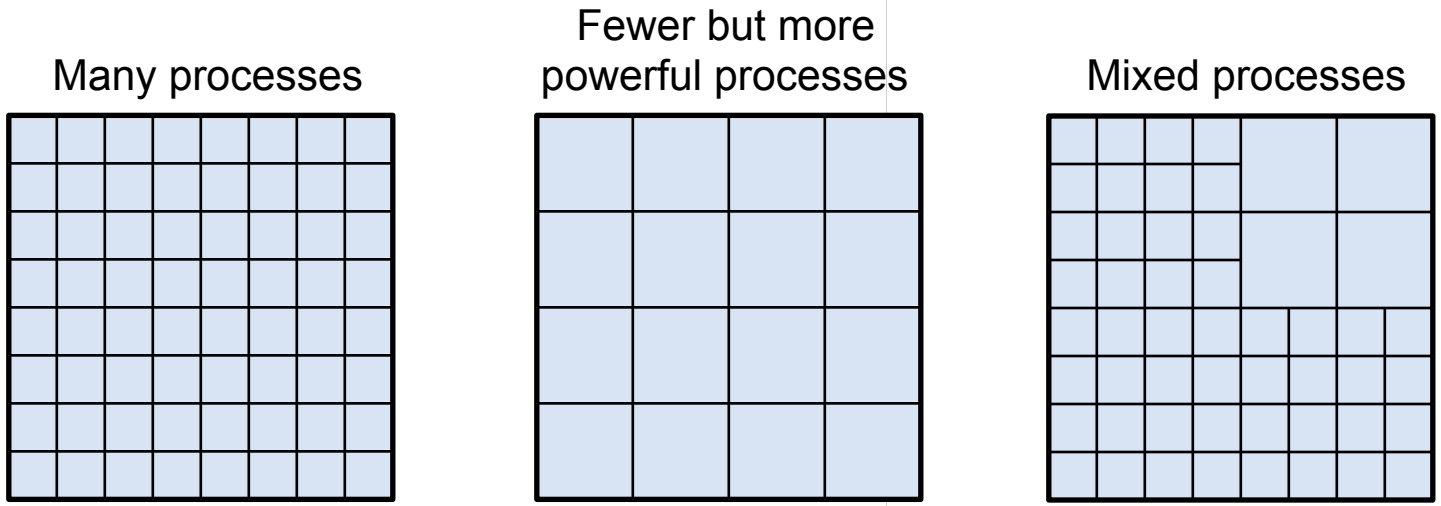
CPU Design Objectives

- A design that is very latency tolerant for processes/threads
- Implement so that each single process can have a good range of performance—as needed at each application time period
- Support process 'swapping' on a clock cycle basis
- Easy data sharing and sharing management
- Do things in a way that support existing applications, but with significantly increased performance and with reasonable opportunities for upgrading those applications
- Plan, with the other architecture ideas, for the total being greater than the sum of the parts—everything works together
- Plan for scalability and with the future in mind

Efficient CPUs: SR-S, SR-P

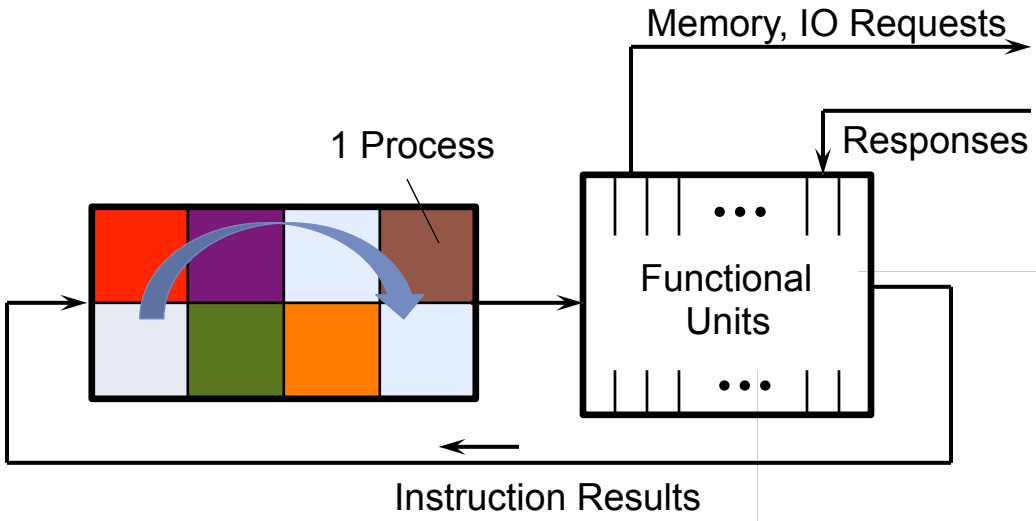
Shared Resource–Serial, Shared Resource–Parallel

- Support multi-threaded implementations so can have CPU efficiencies approaching 100%--Most all execution cycles used
- Capability of many threads in execution; AND have the same hardware support fewer but much more powerful processes, as needed—and change *very* quickly.



Same hardware,
same total compute capability

SR-S Simplified Block Diagram

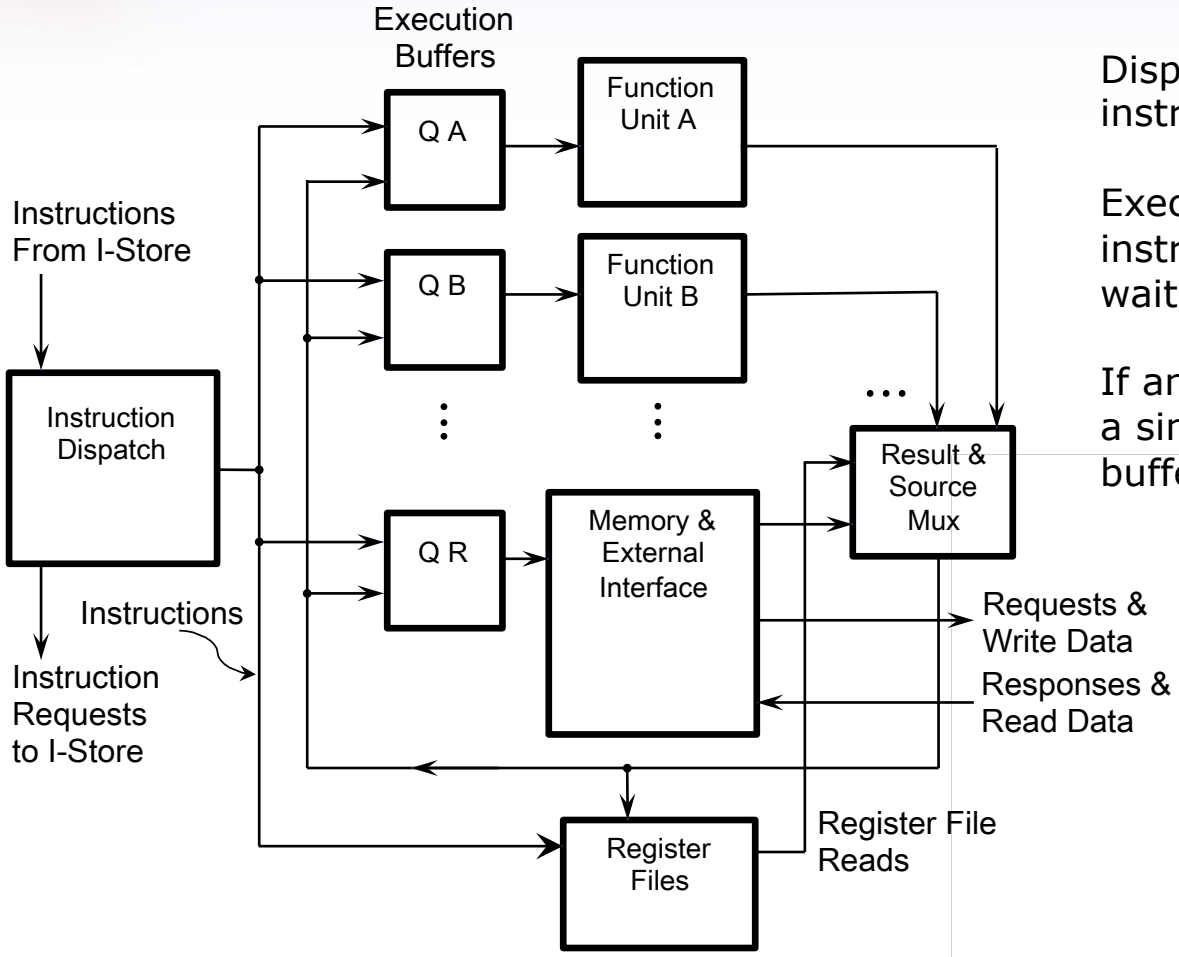


Processes execute round-robin, skipping those inactive. Some functions and capabilities not shown

Colored processes are executing; uncolored processes are ready or are being loaded or unloaded

Starts from the CDC 6600 Peripheral Processor concept, 1964

SR-P Simplified Block Diagram



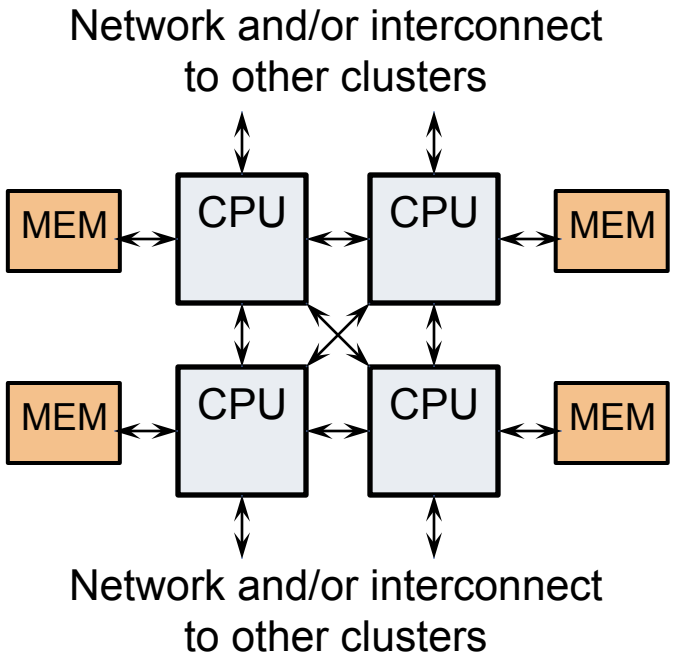
Dispatch interleaves multiple instruction streams as needed

Execution buffers hold instructions for execution, waiting for operands

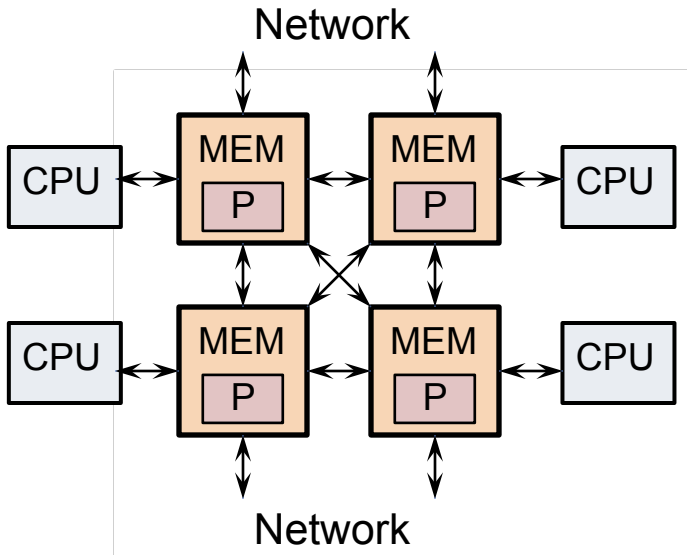
If an instruction is held up, only a single entry in one execution buffer waits

Similar to concepts shown by Bill Dally, then at Nvidia.
There has also been other, previous work.

CPU-centered vs. Memory-centered



The Memory System
IS the Network



Benefits of Memory-centering

- Easier to understand and to manage
- Enables easier implementation of generalized memory functions (Moves, Gather/Scatter for sparse matrix ops, etc.)
- Can provide memory scaling separate from CPUs and CPU boards and packaging
- Enable easier and lower overhead data sharing and data management
- Can offer increased resiliency as needed (like easier checkpoint functionality)
- Basis for network abstraction: everything in a network is addressable—with a single protocol and interface

More at the System Level

- Make a single system-wide abstract communication protocol. CPUs, IO, all memory components (DRAM, Flash/NAND, PCM, FeRAM, …), network functions, etc. use the same interface and protocol. No such thing as an 'IO channel' or 'Memory channel' visible to the network

  Don't send: "Activate, Read, Read, Read, Read, Precharge" to a DRAM

  Send: "Read N bytes"

  The DRAM has intelligence and knows what to do
- Use the example of HMC technology (built-in ECC) and build resiliency into most everything and make error recovery local
- Integrated CPU/GPU heterogeneous nodes have costs as well as benefits, and will raise scaling and overhead issues in the future.

Benefits of the Architectural Features

- Reduced amount of data movement
- Greatly reduced number of interrupts
- Reduced amount of hardware as the higher efficiency gets more done in less space
- Reduced latency in sharing and coherency operations
- Easier parallelism in multiple functions will reduce effort and complexity in use. Much easier fine-grained parallelism
- Significant improvements available in system resiliency
- Increased usage flexibility will enable better performance in a higher percentage of applications

Reduced total energy, increased ease of use, higher performance

And Finally

- What is proposed here is a revolution.
- It is also—and must be—the basis for evolution
- Nothing proposed here automatically breaks current applications. Cache still functions, interrupts still happen, messages get delivered.
- Current applications should actually speed up, even if they take no advantage of the new functions. Examples:
  - Much more memory and memory bandwidth
  - Parallelism is supported more effectively and with lower overhead
  - Processes will not cause cache misses in other processes
- The new functions are independent of each other and can be taken advantage of over time and as needed

Questions and Discussion?